

C Final Test

Task 1	18/20
Task 2	20/20
Task 3	39/40
Task 4	20/20

Compilation penalty (-2.5 <i>per attempted Task</i>)	0/10
--	------

Total:	97/100
--------	--------

```
1: Compiling with gcc -c -Wall everyline.c:
2: Linking everyline.o with gcc -Wall everyline.o testeveryle.o -o ./testeveryle:
3: Compilation and linking Succeeded
4:
5:
6: Running test program ./test-everyline.pl for max 3 seconds
7: debug: cmd=test-everyline.pl
8: [/runprog: normal exit with exit status 0]
9: 20 tests, 20 OK
10: Finished running tests
11:
12:
13: Running valgrind on ./testeveryle
14: > ==2284523== Memcheck, a memory error detector
15: > ==2284523== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
16: > ==2284523== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
17: > ==2284523== Command: ./testeveryle
18: > ==2284523==
19: > testeveryle.c 1: #include <stdio.h>
20: > testeveryle.c 2: #include <stdlib.h>
21: > testeveryle.c 3: #include <assert.h>
22: > testeveryle.c 4:
23: > testeveryle.c 5: #include "everyline.h"
24: > testeveryle.c 6:
25: > testeveryle.c 7:
26: > testeveryle.c 8: static void showline( char *filename, int lineno, char *theline )
27: > testeveryle.c 9: {
28: > testeveryle.c 10:     printf( "%s %4d: %s\n", filename, lineno, theline );
29: > testeveryle.c 11: }
30: > testeveryle.c 12:
31: > testeveryle.c 13:
32: > testeveryle.c 14: int main( void )
33: > testeveryle.c 15: {
34: > testeveryle.c 16:     int n = foreveryline( __FILE__, &showline );
35: > testeveryle.c 17:     printf( "%d lines read\n", n );
36: > testeveryle.c 18:     return 0;
37: > testeveryle.c 19: }
38: > 19 lines read
39: > ==2284523==
40: > ==2284523== HEAP SUMMARY:
41: > ==2284523==     in use at exit: 0 bytes in 0 blocks
42: > ==2284523==   total heap usage: 3 allocs, 3 frees, 8,664 bytes allocated
43: > ==2284523==
44: > ==2284523== All heap blocks were freed -- no leaks are possible
45: > ==2284523==
46: > ==2284523== For lists of detected and suppressed errors, rerun with: -s
47: > ==2284523== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
48: Finished running valgrind
```

Good, no errors or warnings

Good, all tests work

Good, no memory leaks/ uninitialised accesses

```
1: // everyline.[ch] module:
2: // Provided I/O support functions, most notably one to read every line
3: // of a file and invoke a callback with that line.
4:
5: #include <stdio.h>
6: #include <stdlib.h>
7: #include <string.h>
8: #include <assert.h>
9:
10: #include "everyline.h"
11:
12: // int n = foreveryline( filename, eachlinef );
13: // open the given filename, read every line from that a file,
14: // remove the trailing newline (if present) and invoke the given
15: // callback eachlinef with the filename, the line number and the line.
16: // Return the number of lines read, -1 if the file can't be opened.
17: //
18: int foreveryline( char *filename, everylinecb eachlinef )
19: {
20:     FILE *fp = fopen(filename, "r");
21:     if (fp == NULL) {
22:         // File failed to open
23:         return -1;
24:     }
25:
26:     int num_lines = 0;
27:     line buffer;
28:     while (fgets(buffer, MAXLINELEN, fp)) {
29:         num_lines++;
30:         if (buffer[strlen(buffer) - 1] == '\n') {
31:             buffer[strlen(buffer) - 1] = '\0';
32:         }
33:         eachlinef(filename, num_lines, buffer);
34:     }
35:
36:     fclose(fp);
37:
38:     return num_lines;
39: }
40:
```

Good, neat structure

Don't use strlen() repeatedly; call it once, store it's result.
eg int len = strlen(buffer);
- 1 mark

Minor bug: you have ignored the len==0 case (not quite sure if that can happen, but defensive programming suggests that we should defend against it in case it does), if len==0 you access buffer[- 1].
Fix by:

if (len>0 && buffer[len- 1]=='\n') ..

- 1 mark

CFT

test2.txt (1/1)

j1k21

1: Compiling with gcc -c -Wall match.c:
2: Linking match.o with gcc -Wall match.o testmatch.o testutils.o -o ./testmatch:
3: Compilation and linking Succeeded
4:
5:
6: Running test program ./testmatch for max 3 seconds
7: debug: cmd=testmatch
8: 1 tests, 1 OK
9: Finished running tests
10:
11:
12: Running valgrind on ./testmatch
13: > ==2284557== Memcheck, a memory error detector
14: > ==2284557== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
15: > ==2284557== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
16: > ==2284557== Command: ./testmatch
17: > ==2284557==
18: > T match == 'bu': OK
19: > T match == 'bu': OK
20: > T no match as expected: OK
21: > T no match as expected: OK
22: > T no match as expected: OK
23: > T match == 'three': OK
24: > T match == '': OK
25: > T match == '': OK
26: > T match == '': OK
27: > T match == '': OK
28: > T match == '': OK
29: > T match == '': OK
30: > T match == 'x': OK
31: > T match == 'buckle my shoe': OK
32: > ==2284557==
33: > ==2284557== HEAP SUMMARY:
34: > ==2284557== in use at exit: 0 bytes in 0 blocks
35: > ==2284557== total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
36: > ==2284557==
37: > ==2284557== All heap blocks were freed -- no leaks are possible
38: > ==2284557==
39: > ==2284557== For lists of detected and suppressed errors, rerun with: -s
40: > ==2284557== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
41: Finished running valgrind

Good, no errors or warnings

Test suite error, all tests clearly work looking below

Good, no memory leaks/ uninitialised accesses

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <stdbool.h>
4: #include <string.h>
5: #include <assert.h>
6: #include <ctype.h>
7:
8: #include "match.h"
9:
10:
11: // char *wds[] = { "one", "two", NULL };
12: // char *result = matchwords( target, wds );
13: //      Given a target string, and a NULL terminated array of words (each
14: //      word is a char *), attempt to match every word in wds against the
15: //      target, starting the matching process at the start of the target.
16: //      More specifically, you are to:
17: //      - match (and skip) any amount of whitespace (including none) at
18: //      the start of the target,
19: //      - then match wds[0] against the next part of the target exactly as is,
20: //      - then match (and skip) any amount of whitespace (including none),
21: //      - then match wds[1] etc.
22: //      If the target doesn't match all words, return NULL
23: //      If the target matches all words (not necessarily matching the
24: //      whole string), then match and skip any final amount of whitespace
25: //      (including none) after the last word is matched, and return a pointer
26: //      to the first unmatched char beyond that.
27: //
28:
29: char * matchwords( char *target, char **wds )
30: {
31:     char *result = target;
32:
33:     for (char **word = wds; *word; word++) {
34:         while (isspace(*result)) {
35:             result++;
36:         }
37:         for (char *c = *word; *c; c++) {
38:             if (*result++ != *c) {
39:                 return NULL;
40:             }
41:         }
42:     }
43:
44:     while (isspace(*result)) {
45:         result++;
46:     }
47:
48:     return result;
49: }
50:
```

Excellent, very neat



Note: you could use strncmp() but then you need to add strlen(..) to result, actually your code is neater.. excellent

CFT

test3.txt (1/1)j1k21

```
1: Compiling with gcc -c -Wall bst.c:
2: Linking bst.o with gcc -Wall bst.o testbst.o testutils.o -o ./testbst:
3: Compilation and linking Succeeded
4:
5:
6: Running test program ./testbst for max 3 seconds
7: debug: cmd=testbst
8: 1 tests, 1 OK
9: Finished running tests
10:
11:
12: Running valgrind on ./testbst
13: > ==2284579== Memcheck, a memory error detector
14: > ==2284579== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
15: > ==2284579== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
16: > ==2284579== Command: ./testbst
17: > ==2284579==
18: > T hello in? == 0: OK
19: > T there in? == 0: OK
20: > T hi in? == 0: OK
21: > T hello in?#2 == 1: OK
22: > T there in?#2 == 0: OK
23: > T hi in?#2 == 0: OK
24: > T get(hello) == 'bonjour': OK
25: > T hello in?#3 == 1: OK
26: > T there in?#3 == 1: OK
27: > T hi in?#3 == 0: OK
28: > T get(there) == 'alors': OK
29: > T hello in?#4 == 1: OK
30: > T there in?#4 == 1: OK
31: > T hi in?#4 == 0: OK
32: > T goodbye in?#4 == 1: OK
33: > T get(goodbye) == 'ciao': OK
34: > debug: b: { goodbye=>ciao,hello=>bonjour,there=>alors }
35: > debug: foreach gives
36: > debug: kvf: key=goodbye, value=ciao
37: > debug: kvf: key=hello, value=bonjour
38: > debug: kvf: key=there, value=alors
39: > ==2284579==
40: > ==2284579== HEAP SUMMARY:
41: > ==2284579==      in use at exit: 0 bytes in 0 blocks
42: > ==2284579==    total heap usage: 8 allocs, 8 frees, 4,236 bytes allocated
43: > ==2284579==
44: > ==2284579== All heap blocks were freed -- no leaks are possible
45: > ==2284579==
46: > ==2284579== For lists of detected and suppressed errors, rerun with: -s
47: > ==2284579== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
48: Finished running valgrind
```

Good, no errors or warnings

Test suite error - the output below show it worked fine

Good, no memory leaks/ uninitialised accesses

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
4: #include <stdbool.h>
5: #include <assert.h>
6:
7: #include "bst.h"
8:
9:
10: //struct bintree;
11: typedef struct bintree *bintree; // pure binary bintree of key, value pairs
12: struct bintree
13: {
14:     char *key; // the key: a string
15:     void *value; // the value: a generic pointer
16:     bintree left, right; // the left and right subtrees
17: };
18:
19:
20: struct bst
21: {
22:     bst_printkv_func pf; // the (k,v) print function
23:     bst_freev_func ff; // the value free function
24:     bintree t; // the binary bintree of (k,v) pairs itself
25: };
26:
27:
28: // bst d = make_empty_bst( pf, ff );
29: // Create a new empty BST, with an element print function <pf>,
30: // an element free function <ff>, and an empty bintree.
31: // Abort if any memory allocation fails.
32: //
33: bst make_empty_bst( bst_printkv_func pf, bst_freev_func ff )
34: {
35:     bst empty_bst = malloc(sizeof(struct bst));
36:     if (empty_bst == NULL) {
37:         perror("Memory allocation failed.\n");
38:         exit(EXIT_FAILURE);
39:     }
40:
41:     empty_bst->pf = pf;
42:     empty_bst->ff = ff;
43:     empty_bst->t = NULL;
44:
45:     return empty_bst;
46: }
47:
48:
49: // bintree t = makenode( key, value );
50: // Make a new bintree node containing (<key>,<value>),
51: // duplicating <key> via strdup().
52: // Abort if any memory allocation - even the strdup() - fails.
53: //
54: static bintree makenode( char *key, void *value )
55: {
56:     bintree node = malloc(sizeof(struct bintree));
57:     if (node == NULL) {
58:         perror("Memory allocation failed.\n");
59:         exit(EXIT_FAILURE);
60:     }
61:
62:     key = strdup(key);
63:     if (key == NULL) {
64:         free(node);
65:         perror("Memory allocation failed.\n");
66:         exit(EXIT_FAILURE);
67:     }
68:
69:     node->key = key;
70:     node->value = value;
71:     node->left = NULL;
72:     node->right = NULL;
73:
74:     return node;
75: }
76:
77:
78: // add_bst( b, key, value );
79: // Add (<key>,<value>) to <b>. If the given key is already present,
80: // free it's old value and then store <value> instead,
81: // otherwise add a completely new (<key>,<value>) node.
```

Perfect

10/ 10 marks for task 3a

Excellent - like the free(node) in the error case, good cleanup

10/ 10 marks for task 3b

```
82: // The key (but NOT the value) should be duplicated via strdup().
83: // Abort if any memory allocation fails.
84: //
85: void add_bst( bst b, char *key, void *value )
86: {
87:     bintree parent = NULL;
88:     bintree t = b->t;
89:     bool is_left_child = false;
90:
91:     while (t != NULL) {
92:         int cmp = strcmp(key, t->key);
93:         if (cmp == 0) {
94:             break;
95:         }
96:         parent = t;
97:         is_left_child = cmp < 0;
98:         t = is_left_child ? t->left : t->right;
99:     }
100:
101:     if (parent == NULL) {
102:         // The bintree is empty
103:         b->t = makenode(key, value);
104:     } else if (t == NULL) {
105:         // The node does not exist in the bintree
106:         bintree node = makenode(key, value);
107:         if (is_left_child) {
108:             parent->left = node;
109:         } else {
110:             parent->right = node;
111:         }
112:     } else {
113:         // The node exists in the bintree
114:         free(t->value);
115:         t->value = value;
116:     }
117: }
118:
119:
120: // bintree result = find( t, k );
121: // If <k> is not in <t>, return null.
122: // Otherwise return the ptr to the node containing <k> and
123: // it's corresponding value.
124: //
125: static bintree find( bintree t, char *k )
126: {
127:     while( t != NULL )
128:     {
129:         int cmp = strcmp( k, t->key );
130:         if( cmp == 0 ) return t;
131:         t = ( cmp<0 ) ? t->left : t->right;
132:     }
133:     return NULL;
134: }
135:
136:
137: // bool present = in_bst( b, key );
138: // Return true iff <key> is in <b>. Else return false.
139: //
140: bool in_bst( bst b, char *key )
141: {
142:     assert( b != NULL );
143:     bintree t = find( b->t, key );
144:     return t != NULL;
145: }
146:
147:
148: // void * value = get_bst( b, key );
149: // If <key> is in <b>, return the value. Else return NULL.
150: //
151: void * get_bst( bst b, char *key )
152: {
153:     assert( b != NULL );
154:     bintree t = find( b->t, key );
155:     return t != NULL ? t->value : NULL;
156: }
157:
158:
159: // each_bintree( t, &elementfunc, state );
160: // Iterate over every (k,v) pair in <t>, invoking the
161: // given per-kv callback function for each pair
162: // (with the key, the value and the state pointer as parameters)
```

Nice clean iterative structure, with parent and is_left..

Note: personally I'd handle the cmp==0 (key in tree) case right here i.e. move the (bugfixed) final else block here and return afterwards

Minor bug: don't free a value with free(), conditionally invoke b->ff(): - 1 mark

19/ 20 marks for task 3c

```
163: //
164: static void each_bintree( bintree t, bst_kv_func kvf, void *state )
165: {
166:     assert( t != NULL );
167:     if( t->left != NULL ) each_bintree( t->left, kvf, state );
168:     (*kvf)( t->key, t->value, state );
169:     if( t->right != NULL ) each_bintree( t->right, kvf, state );
170: }
171:
172:
173: // foreach_bst( b, &elementfunc, state );
174: // Iterate over every (k,v) pair in <b>, invoking the
175: // given per-element callback function for each pair
176: // (with the key, the value and the state pointer as parameters)
177: //
178: void foreach_bst( bst b, bst_kv_func kvf, void *state )
179: {
180:     assert( b != NULL );
181:     if( b->t != NULL ) each_bintree( b->t, kvf, state );
182: }
183:
184:
185: // int ip = print_bintree( t, pf, out, itemsprinted );
186: // Print the bintree <t> to <out>, as an ordered (key,value) sequence,
187: // separated by commas, and invoking the print function <pf>
188: // for each value. <itemsprinted> is the number of items already printed
189: // Returns the number of items printed after printing this bintree.
190: //
191: static int print_bintree( bintree t, bst_printkv_func pf, FILE *out, int ip )
192: {
193:     assert( t != NULL );
194:     if( t->left != NULL ) ip = print_bintree( t->left, pf, out, ip );
195:     if( ip>0 ) fputc( ',', out );
196:     (*pf)( out, t->key, t->value );
197:     ip++;
198:     if( t->right != NULL ) ip = print_bintree( t->right, pf, out, ip );
199:     return ip;
200: }
201:
202:
203: // print_bst( b, out );
204: // Print the bst <b> to <out>, as an ordered (key,value) sequence,
205: // separated by commas, and invoking the built in print function
206: // for each (key,value) pair.
207: //
208: void print_bst( bst b, FILE *out )
209: {
210:     assert( b != NULL );
211:     assert( b->t != NULL );
212:     fprintf( out, "{ " );
213:     (void) print_bintree( b->t, b->pf, out, 0 );
214:     fprintf( out, " }" );
215: }
216:
217:
218: // free_bintree( t, &freefunc );
219: // free bintree <t>.
220: //
221: static void free_bintree( bintree t, bst_freev_func ff )
222: {
223:     assert( t != NULL );
224:     if( t->left != NULL ) free_bintree( t->left, ff );
225:     if( t->right != NULL ) free_bintree( t->right, ff );
226:     if( ff != NULL ) (*ff)( t->value );
227:     free( t->key ); // was strdup()ed, remember
228:     free( t );
229: }
230:
231:
232: // free_bst( b );
233: // Free the given bst <b>, invoking the free function
234: // for each value.
235: //
236: void free_bst( bst b )
237: {
238:     assert( b != NULL );
239:     assert( b->t != NULL );
240:     free_bintree( b->t, b->ff );
241:     free( b );
242: }
```



```
1: Compiling with gcc -c -Wall analyse.c:
2: Linking analyse.o with gcc -Wall analyse.o testanalyse.o bst.o set.o everyline.o match.o -o ./testanalyse:
3: Compilation and linking Succeeded
4:
5:
6: Running test program ./test-analyse.pl for max 3 seconds
7: debug: cmd=test-analyse.pl
8: debug: read 7 lines from testanalyse, exit status 0
9: debug: mainset: testeveryline.c
10: debug: existset:
analyse.c,analyse.h,bst.c,bst.h,everyline.c,everyline.h,match.c,match.h,queue.h,set.h,testeveryline.c,test
utils.h
11: [/runprog: normal exit with exit status 0]
12: 8 tests, 8 OK
13: Finished running tests
14:
15:
16: Running valgrind on ./testanalyse
17: > ==2284621== Memcheck, a memory error detector
18: > ==2284621== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
19: > ==2284621== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
20: > ==2284621== Command: ./testanalyse
21: > ==2284621==
22: > mainset: testeveryline.c
23: > existset:
analyse.c,analyse.h,bst.c,bst.h,everyline.c,everyline.h,match.c,match.h,queue.h,set.h,testeveryline.c,test
utils.h
24: > c2inc(analyse.c): analyse.h,bst.h,everyline.h,match.h,set.h
25: > c2inc(bst.c): bst.h
26: > c2inc(everyline.c): everyline.h
27: > c2inc(match.c): match.h
28: > c2inc(testeveryline.c): everyline.h
29: > ==2284621==
30: > ==2284621== HEAP SUMMARY:
31: > ==2284621==      in use at exit: 0 bytes in 0 blocks
32: > ==2284621==    total heap usage: 78 allocs, 78 frees, 94,546 bytes allocated
33: > ==2284621==
34: > ==2284621== All heap blocks were freed -- no leaks are possible
35: > ==2284621==
36: > ==2284621== For lists of detected and suppressed errors, rerun with: -s
37: > ==2284621== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
38: Finished running valgrind
```

Good, no errors or warnings

Good, all tests work

Good, no memory leaks/ uninitialised accesses

```
1: // analyse.[ch]: find all includes and main() in *.ch files and build
2: //     some data structures to represent what we find
3:
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include <stdbool.h>
7: #include <string.h>
8: #include <assert.h>
9: #include <glob.h>
10:
11: #include "bst.h"
12: #include "set.h"
13: #include "analyse.h"
14: #include "everyline.h"
15: #include "match.h"
16:
17: static analysis a;
18:
19: static void recordmain( char *filename )
20: {
21:     // printf( "debug: %s contains main()\n", filename );
22:     // TASK 4: build mainset.
23:
24:     add_set(a->mainset, filename);
25: }
26:
27:
28: static void recordfileexists( char *filename )
29: {
30:     // printf( "debug: file %s exists\n", filename );
31:     // TASK 4: build existset.
32:
33:     add_set(a->existset, filename);
34: }
35:
36:
37: static void recordinclude( char *filename, char *oneinc )
38: {
39:     // printf( "debug: %s directly includes %s\n", filename, oneinc );
40:     // TASK 4: build c2inc.
41:
42:     bool src_in_bst = in_bst(a->c2inc, filename);
43:
44:     set value;
45:
46:     if (src_in_bst) {
47:         value = (set) get_bst(a->c2inc, filename);
48:     } else {
49:         value = make_set();
50:         add_bst(a->c2inc, filename, value);
51:     }
52:
53:     add_set(value, oneinc);
54: }
55:
56:
57: static char *include[] = { "#include", NULL };
58: static char *hasmain[] = { "int", "main", "(", NULL };
59:
60:
61: static void examineline( char *filename, int ln, char *line )
62: {
63:     //printf( "debug: examine line %d, %s\n", ln, line );
64:
65:     char *inc = matchwords( line, include );
66:     if( inc != NULL && *inc == ' ' )
67:     {
68:         inc++;
69:         char *lastquote = strchr(inc, '"');
70:         if( lastquote != NULL ) *lastquote = '\0';
71:         recordinclude( filename, inc );
72:     }
73:     char *main = matchwords( line, hasmain );
74:     if( main != NULL )
75:     {
76:         recordmain( filename );
77:     }
78: }
79:
80:
81: static void print_wrapper( FILE *out, char *key, void *value )
```

Good

Good

Good

Great

Minor note: Don't need to use in_bst() and get_bst(), or have the bool var, get_bst() returns NULL if the key is missing. Write:

```
set value = get_bst(...);
if( value==NULL ) {
    value = make_set();
    add_bst(...);
}
add_set(...);
```

```
82: {
83:     fprintf( out, "%s=>", key );
84:     print_set( value, out ); // value is itself a set
85: }
86:
87:
88: static void free_wrapper( void *value )
89: {
90:     free_set( (set) value );
91: }
92:
93:
94: static analysis make_analysis( void )
95: {
96:     a = malloc(sizeof(struct analysis));
97:     assert(a != NULL);
98:     a->existset = make_set();
99:     a->mainset = make_set();
100:    a->c2inc = make_empty_bst(&print_wrapper, &free_wrapper);
101:    return a;
102: }
103:
104:
105: analysis analyse( void )
106: {
107:     a = make_analysis();
108:
109:     glob_t globbuf;
110:     glob( "*.ch", 0, NULL, &globbuf );
111:     int nfiles = globbuf.gl_pathc; // How many paths matched
112:
113:     for( int fileno=0; fileno<nfiles; fileno++ )
114:     {
115:         char *filename = globbuf.gl_pathv[fileno];
116:         assert( filename != NULL );
117:         recordfileexists( filename );
118:         foreveryline( filename, &examineline );
119:     }
120:     globfree( &globbuf );
121:     return a;
122: }
123:
124:
125: void free_analysis( analysis a )
126: {
127:     if( a->mainset != NULL ) free_set( a->mainset );
128:     if( a->existset != NULL ) free_set( a->existset );
129:     if( a->c2inc != NULL ) free_bst( a->c2inc );
130:     free( a );
131: }
```

Fine

Good